Relational Algebra

1

Relational Query Languages

- *Query languages*: Allow manipulation and retrieval of data from a database.
- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on logic.
  - Allows for much optimization.
- Query Languages != programming languages!
  - QLs not expected to be "Turing complete".
  - QLs not intended to be used for complex calculations.
  - QLs support easy, efficient access to large data sets.

2

Formal Relational Query Languages

Two mathematical Query Languages form the basis for "real" languages (e.g. SQL), and for implementation:

❶ *Relational Algebra*: More operational, very useful for representing execution plans.

❷ *Relational Calculus*: Lets users describe what they want, rather than how to compute it. (Non-operational, *declarative*.)

☛ *Understanding Algebra & Calculus is key to*
☛ *understanding SQL, query processing!*

3

Preliminaries

- A query is applied to *relation instances*, and the result of a query is also a relation instance.
  - *Schemas* of input relations for a query are fixed (but query will run regardless of instance!)
  - The schema for the *result* of a given query is also fixed! Determined by definition of query language constructs.
- Positional vs. named-field notation:
  - Positional notation easier for formal definitions, named-field notation more readable.
  - Both used in Relational Algebra and SQL

4

Example Instances

- "Sailors" and "Reserves" relations for our examples.
- We'll use positional or named field notation, assume that names of fields in query results are `inherited' from names of fields in query input relations.

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

5

Relational Algebra

- Basic operations:
  - *Selection* (σ) Selects a subset of rows from relation.
  - *Projection* (π) Deletes unwanted columns from relation.
  - *Cross-product* ($\times$) Allows us to combine two relations.
  - *Set-difference* ($-$) Tuples in reln. 1, but not in reln. 2.
  - *Union* ($\cup$) Tuples in reln. 1 and in reln. 2.
- Additional operations:
  - Intersection, *join*, division, renaming: Not essential, but (very!) useful.
- Since each operation returns a relation, operations can be *composed*! (Algebra is "closed".)

6

Projection

- Deletes attributes that are not in *projection list*.
- *Schema* of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- Projection operator has to eliminate *duplicates*! (Why??)
  - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Why not?)

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$\pi_{sname,rating}(S2)$

age
35.0
55.5

$\pi_{age}(S2)$

7

Selection

- Selects rows that satisfy *selection condition*.
- No duplicates in result! (Why?)
- *Schema* of result identical to schema of (only) input relation.
- *Result* relation can be the *input* for another relational algebra operation! (*Operator composition*.)

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$\sigma_{rating>8}(S2)$

sname	rating
yuppy	9
rusty	10

$\pi_{sname,rating}(\sigma_{rating>8}(S2))$

8

Union, Intersection, Set-Difference

- All of these operations take two input relations, which must be <u>union-compatible</u>:
  - Same number of fields.
  - `Corresponding' fields have the same type.
- What is the *schema* of result?

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

sid	sname	rating	age
22	dustin	7	45.0

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

9

Cross-Product

- Each row of S1 is paired with each row of R1.
- *Result schema* has one field per field of S1 and R1, with field names `inherited' if possible.
  - *Conflict*: Both S1 and R1 have a field called *sid*.

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

☛ <u>*Renaming operator*</u>: $\rho(C(1 \to sid1, 5 \to sid2), S1 \times R1)$

10

Joins

- <u>*Condition Join*</u>: $R \bowtie_C S = \sigma_C (R \times S)$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

$S1 \bowtie_{S1.sid < R1.sid} R1$

- *Result schema* same as that of cross-product.
- Fewer tuples than cross-product, might be able to compute more efficiently
- Sometimes called a *theta-join*.

11

Joins

- <u>*Equi-Join*</u>: A special case of condition join where the condition *c* contains only ***equalities*** and ^.

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

$S1 \bowtie_{sid} R1$

- *Result schema* similar to cross-product, but only one copy of fields for which equality is specified.
- <u>*Natural Join*</u>: Equijoin on *all* common fields.

12

Find names of sailors who've reserved boat #103

- Solution 1: $\pi_{sname}((\sigma_{bid=103} Reserves) \bowtie Sailors)$
- Solution 2: $\rho\ (Temp1,\ \sigma_{bid=103} Reserves)$

  $\rho\ (Temp2,\ Temp1 \bowtie Sailors)$

  $\pi_{sname}(Temp2)$
- Solution 3: $\pi_{sname}(\sigma_{bid=103}(Reserves \bowtie Sailors))$

13

Find names of sailors who've reserved a red boat

- Information about boat color only available in Boats; so need an extra join:

$\pi_{sname}((\sigma_{color='red'} Boats) \bowtie Reserves \bowtie Sailors)$

- A more efficient solution:

$\pi_{sname}(\pi_{sid}((\pi_{bid}\sigma_{color='red'} Boats) \bowtie Res) \bowtie Sailors)$

☛ *A query optimizer can find this given the first solution!*

14

Find sailors who've reserved a red or a green boat

- Can identify all red or green boats, then find sailors who've reserved one of these boats:

$\rho\ (Tempboats, (\sigma_{color='red' \vee color='green'} Boats))$

$\pi_{sname}(Tempboats \bowtie Reserves \bowtie Sailors)$

- Can also define Tempboats using union! (How?)
- What happens if is replaced by in this query?

15

Find sailors who've reserved a red and a green boat

- Previous approach won't work! Must identify sailors who've reserved red boats, sailors who've reserved green boats, then find the intersection (note that *sid* is a key for Sailors):

$\rho\ (Tempred, \pi_{sid}((\sigma_{color='red'} Boats) \bowtie Reserves))$

$\rho\ (Tempgreen, \pi_{sid}((\sigma_{color='green'} Boats) \bowtie Reserves))$

$\pi_{sname}((Tempred \cap Tempgreen) \bowtie Sailors)$

16

Relational Calculus

17

Relational Calculus

- Comes in two flavors: <u>*Tuple relational calculus*</u> (TRC) and <u>*Domain relational calculus*</u> (DRC).
- Calculus has *variables, constants, comparison ops, logical connectives,* and *quantifiers*.
  - <u>*TRC*</u>: Variables range over (i.e., get bound to) *tuples*.
  - <u>*DRC*</u>: Variables range over *domain elements* (= field values).
  - Both TRC and DRC are simple subsets of first-order logic.
- Expressions in the calculus are called *formulas*. An answer tuple is essentially an assignment of constants to variables that make the formula evaluate to *true*.

18

Tuple Relational Calculus

- *Query* has the form: { T | p(T)}
- *Answer* includes all tuples T that make the *formula* p(T) be *true*.
- *Formula* is recursively defined, starting with simple *atomic formulas* (getting tuples from relations or making comparisons of values), and building bigger and better formulas using the *logical connectives*.

19

TRC Formulas

- *Atomic formula:*
  - R ∈ Rel, or R.a *op* S.b, or R.a *op* constant
  - *op* is one of $<, >, =, \leq, \geq, \neq$
- *Formula:*
  - an atomic formula, or
  - $\neg p, p \wedge q, p \vee q$, where p and q are formulas, or
  - $\exists X\,(p(X))$, where variable X is *free* in p(X), or
  - $\forall X\,(p(X))$, where variable X is *free* in p(X)

20

Free and Bound Variables

- The use of quantifiers ∀X and ∃X in a formula is said to *bind* X.
  - A variable that is not bound is free.
- Let us revisit the definition of a query: {T | p(T)}
- There is an important restriction: the variable T that appears to the left of `|' must be the *only* free variable in the formula p(...).

21

Find all sailors with a rating above 7

- {S | S ∈ Sailors ^ S.rating > 7}
- Query is evaluated on an instance of Sailors
- Tuple variable S is instantiated to each tuple of this instance in turn, and the condition "S.rating > 7" is applied to each such tuple.
- Answer contains all instances of S (which are tuples of Sailors) satisfying the condition.

22

Find sailors rated > 7 who've reserved boat #103

- {S | (S ∈ Sailors) ^ (S.rating > 7) ^ (∃ R ∈ Reserves (R.sid = S.sid ^ R.bid = 103))}
- Note the use of ∃ to find a tuple in Reserves that `joins with' the Sailors tuple under consideration.
- R is bound, S is not

23

Unsafe Queries, Expressive Power

- It is possible to write syntactically correct calculus queries that have an infinite number of answers! Such queries are called *unsafe*.
  - e.g., $\{S \mid \neg\{S \in Sailors\}\}$
- It is known that every query that can be expressed in relational algebra can be expressed as a safe query in DRC / TRC; the converse is also true.
- *Relational Completeness*: Query language (e.g., SQL) can express every query that is expressible in relational algebra / calculus.

24

Summary

- The relational model has rigorously defined query languages that are simple and powerful.
- Relational algebra is more operational; useful as internal representation for query evaluation plans.
- Relational calculus is non-operational, and users define queries in terms of what they want, not in terms of how to compute it. (*Declarativeness.*)
- Several ways of expressing a given query; a *query optimizer* should choose the most efficient version.
- Algebra and safe calculus have same *expressive power*, leading to the notion of *relational completeness.*

25

Nested Relations

- Attributes can be scalar (as before) or <u>relation-valued</u>
- Definition is <u>recursive</u>
- <u>*Example*</u>:
  create table Book (title: string, author:string,
  copies: (publ: string,
  pages: integer,
  date: integer))
- "copies" is a relation-valued field

26

Nested Relational Algebra

- A spectrum of algebras can be defined
- At one end:
  - Relational algebra plus <u>*nest*</u> (ν) and <u>*unnest*</u> (μ):
    If B =

title	author	copies		
		publ	pages	date
Moby Dick	Melville	Prentice Hall	613	1971
		McGraw Hill	542	1942
Marmion	Scott	{ }		

27

Nesting and Unnesting

- … then μ (B, copies) =

title	author	publ	pages	date
Moby Dick	Melville	Prentice Hall	613	1971
Moby Dick	Melville	McGraw Hill	542	1942
Marmion	Scott	*null*	*null*	*null*

- Nulls must be supported *in algebra*
- ν (μ (B, copies), copies (publ, pages, date)) = B
- ν, μ inverses iff S $\rightarrow$ N
  - S is set of scalar fields
  - N is set of non-scalar fields
  - This is called PNF (partitioned normal form)

28

Extending Relational Operators

- At other end of spectrum:
  - *Selection* allows set comparators and constants and use of select, project inside the formula
  - *Projection* allows arbitrary NF2 algebra expression in addition to simple field names
  - *Union, difference*: recursive definitions
  - *Cross product*: usually just relational.
- Example: retrieve title, number of pages of all books by Melville:
  - π[title, π[pages](copies)](σ[author='Melville'](B))

29

Nested Relations Summary

- An early step on the way to OODBMS
- No products, only prototypes, but:
  - Many ideas from NF2 relations have survived
  - Collection types in SQL3 (nesting, unnesting)
  - Algebra ideas useful for Object Database QP
- Can provide a more natural model of data
- Are a straightforward extension of relations:
  - many solutions are thus also straightforward
  - formal foundation of relational model remains

30